Module 3 – Window Functions

Overview

The main purpose of this lab is to familiarise yourself with window functions for aggregations and ranking within TSQL.

- Aggregations using OVER, PARTITION BY and ORDER BY clauses
- Ranking using RANK(), DENSE_RANK(), ROW_NUMBER() and NTILE(x)

Objectives

At the end of this lab, you will be able to:

- Select the clauses required in window functions
- Select the correct ranking function for your requirements

Setup: Launch SQL Server Management Studio

1. On the Start menu, click All Programs, click Microsoft SQL Server 2014 and then click SQL Server Management Studio.
2. The Microsoft SQL Server Management Studio window opens, and then the Connect to Server dialog box will appear.
3. In the Connect to Server dialog box, click Connect to accept the default settings.
4. On the toolbar, click New Query, and either select the QATSQLPLUS database in the Available Databases box, or type USE QATSQLPLUS in the query window.
5. If you wish, you may save your queries to My Documents or the desktop. All modules are separate and you will not require any queries from this module in any later module.

Exercise 1: Stock movement

In this exercise, you will create queries to return the current stock and stock movement history for all the books using the BookTransfers table.

The main tasks for this exercise are as follows:

1. Create a query to review the content of the dbo.BookTransfers table.
2. Create a query to return the current stock for each book.
3. Create a query to return the stock movement history, showing each transfers and how that affected the stock held.

Task 1 : Review table content

1. Write a query to return columns from the dbo.BookTransfers table. The columns returned should be:
   - ProductID
   - TransferDate
   - TransferAmount
2. Test the query. The query should return 45 rows with both positive (received) and negative (sent) values in the TransferAmount column.
3. Keep the query window open for the following tasks.

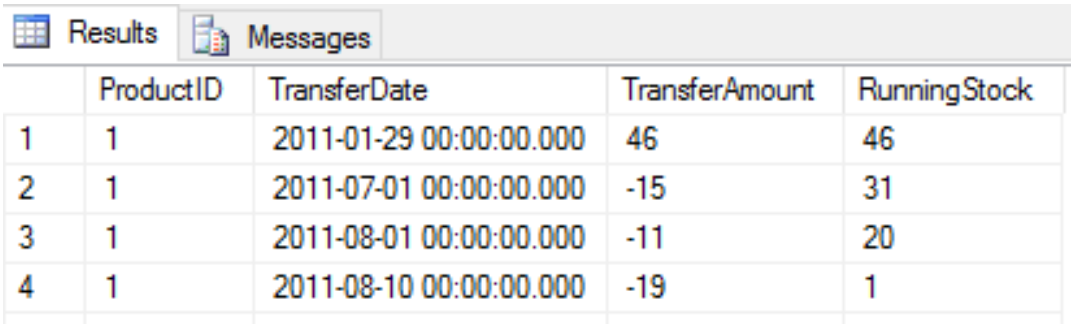
Task 2 : Current stock

1. Write a query to aggregate TransferAmount within the dbo.BookTransfers table. The columns returned should be:
   - ProductID
   - SUM(TransferAmount) aliased as Stock
2. Test the query. The query should return 8 rows holding Stock values between 0 and 139.
3. Keep the query window open for the following tasks.

Task 3: Movement History

1. Write a query to show the running total of stock from the opening stock to last transaction, ordered by date, similar to a bank statement. The RunningStock for each ProductID should be kept separate. The columns returned should be:
   - ProductID
   - TransferDate
   - TransferAmount
   - RunningStock (using the SUM function)

2. Test the query. The query should return 46 rows (as per task 1). The first product results are shown below, although all products should be returned.



Results Messages

	ProductID	TransferDate	TransferAmount	RunningStock
1	1	2011-01-29 00:00:00.000	46	46
2	1	2011-07-01 00:00:00.000	-15	31
3	1	2011-08-01 00:00:00.000	-11	20
4	1	2011-08-10 00:00:00.000	-19	1

3. Keep the query window open for the following tasks.

Exercise 2: League Tables

In this exercise, you find produce a league table of the courses by number of delegates.

The main tasks for this exercise are as follows:

1. Create a query that returns the list of courses and the league position using available ranking functions.
2. Create a query that returns the top course for each Vendor.

Task 1 : League table

1. Write a query to return all columns from the dbo.VendorCourseDelegateCount view. The columns returned should be:
   - VendorName
   - CourseName
   - NumberDelegates
2. Test the query. The query should return 7 rows.
3. Add an additional columns to the query to return the league positions using the 4 ranking functions listed below using the NumberOfDelegates descending to order the courses:
   - RANK()
   - DENSE_RANK()
   - ROW_NUMBER()
   - NTILE(3)
4. Test the query again. The first 3 additional columns should show the league positions with differences in handling equal values. The last column (NTILE) will cut the list into 3 similar sized blocks.
5. Keep the query window open for the following tasks.

Task 2 : League leaders

1. Write a query to return all columns from the dbo.VendorCourseDelegateCount view. The columns returned should be:
   - VendorName
   - CourseName
   - NumberDelegates
2. Test the query. This should return 7 rows.
3. Add a league position column to the query using the RANK() function to return the position within the Vendor. Alias the new column as League_Pos.
4. Test the query. This should return 7 rows. Each Vendor should return its own league table starting at 1.
5. Using either a CTE or derived table, alter the query to only return the course with League_Pos = 1.
6. Test the query. This should return 3 rows. Each vendor should have a top attended course.
7. The query window can be closed. Save the query if you wish.

Answers

The answers below are for example only. Coding style and order of columns should not matter.

Ex 1 Task 1

```
--Task 1:
SELECT ProductID, TransferDate, TransferAmount
  FROM dbo.BookTransfers

--Task 2:
SELECT ProductID, SUM(TransferAmount) AS Stock
  FROM dbo.BookTransfers
  GROUP BY ProductID

--Task 3:
SELECT ProductID, TransferDate, TransferAmount,
  SUM(TransferAmount) OVER
  (PARTITION BY ProductID ORDER BY TransferDate)
    AS RunningStock
    FROM dbo.BookTransfers
```

Ex 2 Task 1

```
--Task 1a:
SELECT *
  FROM dbo.VendorCourseDelegateCount

--Task 1b:
SELECT *,
  RANK() OVER (ORDER BY NumberDelegates DESC)
    AS LeaguePos_Rank,
  DENSE_RANK() OVER (ORDER BY NumberDelegates DESC)
    AS LeaguePos_DenseRank,
  ROW_NUMBER() OVER (ORDER BY NumberDelegates DESC)
    AS LeaguePos_Row,
  NTILE(3) OVER (ORDER BY NumberDelegates DESC)
    AS LeaguePos_Ntile
  FROM dbo.VendorCourseDelegateCount
GO
```

Ex 2 Task 2

```
--Task 2a:
SELECT *
  FROM dbo.VendorCourseDelegateCount
GO


--Task 2b:
SELECT *, RANK() OVER (PARTITION BY Vendorname ORDER BY NumberDelegates
DESC)
  FROM dbo.VendorCourseDelegateCount
GO


--Task 2c:
-- Using CTE
WITH Ranked_Courses AS (
  SELECT *,
    RANK() OVER (PARTITION BY VendorName
      ORDER BY NumberDelegates DESC) AS LeaguePos
    FROM dbo.VendorCourseDelegateCount
)
SELECT VendorName, CourseName, NumberDelegates
  FROM Ranked_Courses
  WHERE LeaguePos = 1
GO


-- Using Derived Table
SELECT VendorName, CourseName, NumberDelegates
  FROM (
    SELECT *,
       RANK() OVER (PARTITION BY VendorName
        ORDER BY NumberDelegates DESC) AS LeaguePos
       FROM dbo.VendorCourseDelegateCount
  ) AS DT
  WHERE LeaguePos = 1
GO
```